PROGRAM 1

Write a C++ program to find largest, smallest & second largest of three numbers using inline functions MAX & Min

```
#include<iostream>
using namespace std;
inline int Max(int a,int b)
{

   return(a>b)?a:b;
}
inline int Min(int a, int b)
{

   return(a<b)?a:b;
}
void findLargestSmallest(int arr[], int size)
{
   int largest=arr[0];
   int smallest=arr[0];
   int secondlargest=arr[0];
   for(int i=1;i<size;i++)
   {

      largest=Max(largest,arr[i]);
      smallest=Min(smallest,arr[i]);
   }
   for(int i=1;i<size;i++)
   {
      if(arr[i]!=largest)
```

```
        {
            secondlargest=Max(secondlargest,arr[i]);
        }
    }
    cout<<"Largest number is"<<largest<<std::endl;
    cout<<"Smallest number is"<<smallest<<std::endl;
    cout<<"Second Largest number is"<<secondlargest<<std::endl;

}
int main()
{

    int size;
    cout<<"Enter the size of the array\n";
    cin>>size;
    int arr[size];
    cout<<"Enter"<<size<<"numbers:\n";
    for(int i=0;i<size;i++)
    {
        std::cin>>arr[i];
    }
    findLargestSmallest(arr,size);
    return 0;
}
```

OUTPUT:

```
Enter the size of the array
4
Enter4numbers:
23 32 78 1
Largest number is78
Smallest number is1
Second Largest number is32
```

PROGRAM 2:

Define a STUDENT class with USN, Name & Marks in 3 tests of a subject. Declare an array of 10 STUDENT objects. Using appropriate functions, find the average of the two better marks for each student. Print the USN, Name & the average marks of all the students

```
#include<iostream>
#include<string>
using namespace std;
class STUDENT
{
private:
  string usn;
  string name;
  int marks[3];
public:
  void inputDetails()
  {
    cout<<"Enter USN:";
    cin>>usn;
    cout<<"Enter Name:";
    cin.ignore();
    getline(cin, name);
    cout<<"Enter Marks obtained in 3 tests:";
```

```
    for(int i=0; i<3; i++)
    {
        cin>>marks[i];
    }
}
float calculateAverage()
{
    for(int i=0; i<2; i++)
    {

        for(int j=0; j<2-i; j++)
            {
                if(marks[j]<marks[j+1])
                {
                    int temp=marks[j];
                    marks[j]=marks[j+1];
                    marks[j+1]=temp;
                }
            }
    }
    return(marks[0]+marks[1])/2.0;
}
void displayDetails(float average)
{

    cout<<"USN:"<<usn<<endl;
    cout<<"Name:"<<name<<endl;
    cout<<"Average Marks:"<<average<<endl;
    cout<<endl;
```

```
    }
};
int main()
{
    STUDENT students[10];
    for(int i=0; i<1; i++)
    {
        cout<<"Enter the details of the student"<<i+1<<endl;
        students[i].inputDetails();
        cout<<endl;
    }
    for(int i=0; i<1; i++)
    {
        float average=students[i].calculateAverage();
        students[i].displayDetails(average);
    }
    return 0;
}
```

OUTPUT:

```
Enter the details of the student1
Enter USN:1DA21EC001
Enter Name:ANAND
Enter Marks obtained in 3 tests:23 24 25

USN:1DA21EC001
Name:ANAND
Average Marks:24.5
```

PROGRAM 3:

Write a C++ program to create class called MATRIX using two-dimensional array of integers, by overloading the operator == which checks the compatibility of two matrices to be added and subtracted. Perform the addition and subtraction by overloading + and – operators respectively. Display the results by overloading the operator

```
#include<iostream>
#include<vector>
using namespace std;
class MATRIX
{
private:
   vector<vector<int>>matrix;
   int rows;
   int columns;
public:
   MATRIX(int numRows, int numColumns)
   {
      rows=numRows;
      columns=numColumns;
      matrix.resize(rows, vector<int>(columns));
   }
   void inputMatrix()
   {
      cout<<"Enter the elements of the matrix:"<<endl;
      for(int i=0; i<rows; i++)
      {
         for(int j=0; j<columns;j++)
         {
            cin>>matrix[i][j];
```

```
        }
      }
    }
    bool operator==(const MATRIX& otherMatrix)const
    {
      return(rows==otherMatrix.rows&&columns==otherMatrix.columns);
    }
    MATRIX operator+(const MATRIX& otherMatrix)const
    {
      MATRIX result(rows,columns);
      if(*this==otherMatrix)
      {
        for(int i=0;i<rows; i++)
        {
          for(int j=0; j<columns; j++)
        {
          result.matrix[i][j]=matrix[i][j]+otherMatrix.matrix[i][j];
        }
      }
    }
    else
    {
      cout<<"Error: Matrix are not compatible for adddition!"<<endl;
    }
    return result;
}
MATRIX operator-(const MATRIX& otherMatrix)const
    {
      MATRIX result(rows,columns);
```

```
        if(*this==otherMatrix)
        {
            for(int i=0;i<rows; i++)
            {
                for(int j=0; j<columns; j++)
            {
                result.matrix[i][j]=matrix[i][j]-otherMatrix.matrix[i][j];
            }
        }
    }
    else
    {
        cout<<"Error: Matrix are not compatible for  subtraction!"<<endl;
    }
    return result;
    }
friend ostream& operator<<(ostream& os, const MATRIX&matrix)
{
    for(int i=0; i<matrix.rows; i++)
    {
        for(int j=0; j<matrix.columns; j++)
        {
            os<<matrix.matrix[i][j]<<" ";
        }
        os<<endl;
    }
    return os;
}
};
```

```
int main()
{
   int rows, columns;
   cout<<"Enter the size of matrix 1:";
   cin>>rows>>columns;
   cout<<"Enter the size of matrix 2:";
   cin>>rows>>columns;
   MATRIX m1(rows,columns);
   MATRIX m2(rows,columns);
   if(m1==m2)
   {
   cout<<"Enter the elements for matrix 1:"<<endl;
   m1.inputMatrix();
   MATRIX m2(rows,columns);
   cout<<"Enter the elements for matrix 2:"<<endl;
   m2.inputMatrix();
      MATRIX m3=m1+m2;
      cout<<"Addition Result:"<<endl;
      cout<<m3<<endl;
      MATRIX m4=m1-m2;
      cout<<"Subtraction Result:"<<endl;
      cout<<m4<<endl;
   }
   else
   {
      cout<<"Error in the order of matrices. so enable to compute addition and subtraction of matrices";
   }
   return 0;
}
```

OUTPUT:

```
Enter the size of matrix 1:2 2
Enter the size of matrix 2:2 2
Enter the elements for matrix 1:
Enter the elements of the matrix:
1 2
3 4
Enter the elements for matrix 2:
Enter the elements of the matrix:
3 4
5 6
Addition Result:
4 6
8 10

Subtraction Result:
-2 -2
-2 -2
```

PROGRAM 4:

Write a C++ program to explain virtual function (Polymorphism) by creating a base class polygon which has virtual function areas two classes rectangle & triangle derived from polygon & they have area to calculate & return the area of rectangle & triangle respectively

```
#include<iostream>
using namespace std;
class POLYGON
{
public:
   virtual double area() const=0;
};
class RECTANGLE: public POLYGON
{
private:
   double length;
```

```
    double width;
public:
    RECTANGLE(double l, double w): length(l),width(w){}
    double area() const override
    {
        return length*width;
    }
};
class TRIANGLE: public POLYGON
{
private:
    double base;
    double height;
public:
    TRIANGLE(double b, double h): base(b), height(h){}
    double area() const override
    {
        return 0.5*base*height;
    }
};
int main()
{
    POLYGON* poly;
    RECTANGLE rect(5.0,3.0);
    TRIANGLE tri(4.0,6.0);
    poly=&rect;
    cout<<"Area of Rectangle:"<<poly->area()<<endl;
    poly=&tri;
    cout<<"Area of Triangle:"<<poly->area()<<endl;
```

```
    return 0;
}
```

OUTPUT:

```
Area of Rectangle:15
Area of Triangle:12
```

PROGRAM 5:

Demonstrate simple inheritance concept by creating a base class FATHER with data members: First Name, Surname, DOB & bank Balance and creating a derived class SON, which inherits: Surname & Bank Balance feature from base class but provides its own feature: First Name & DOB. Create & initialize F1 & S1 objects with appropriate constructors & display the FATHER & SON details.

```
#include<iostream>
#include<string>
using namespace std;
class FATHER
{
protected:
    string surname;
    double bankBalance;
public:
    FATHER(const string&s, double balance):surname(s),bankBalance(balance){}
    void displayFatherDetails()
    {
        cout<<"Father's Surname:"<<surname<<endl;
        cout<<"Father's Bank Balance:$"<<bankBalance<<endl;
    }
};
class SON:public FATHER
```

```
{
private:
   string firstName;
   string dob;
public:
   SON(const string&f, const string&d, const string&s, double
balance):FATHER(s,balance),firstName(f),dob(d){}
   void displaySonDetails()
   {
      cout<<"Son's First Name:"<<firstName<<endl;
      cout<<"Son's DOB:"<<dob<<endl;
      cout<<"Son's  Surname:"<<surname<<endl;
      cout<<"Son's Bank Balance:$"<<bankBalance<<endl;
   }
};
int main()
{
   FATHER F1("Smith", 100000.0);
   SON S1("John","2001-05-10","Smith",500.0);
   cout<<"Father's Details:"<<endl;
   F1.displayFatherDetails();
   cout<<endl;
   cout<<"Son's Details:"<<endl;
   S1.displaySonDetails();
   return 0;

}
```

OUTPUT:

```
Father's Details:
Father's Surname:Smith
Father's Bank Balance:$100000

Son's Details:
Son's First Name:John
Son's DOB:2001-05-10
Son's  Surname:Smith
Son's Bank Balance:$500
```

PROGRAM 6:

Write a C++ program to accept the student detail such as name & 3 different marks by get_data() method & display the name & average of marks using display() method. Define a friend function for calculating the average marks using the method mark_avg().

```
#include<iostream>
using namespace std;
class STUDENT
{
private:
   string name;
   int marks[3];
public:
   void get_data()
   {
      cout<<"Enter the student's name:";
      getline(cin>>ws, name);
      cout<<"Enter the marks for three subjects:"<<endl;
      for(int i=0;i<3;i++)
      {
         cout<<"Enter marks"<<i+1<<":";
```

```
            cin>>marks[i];
        }

    }
    void display()
    {
        cout<<"Student Name:"<<name<<endl;
    }
    friend float mark_avg(const STUDENT& student);
};
float mark_avg(const STUDENT& student)
{
    int sum=0;
    for(int i=0;i<3;i++)
    {
        sum+=student.marks[i];
    }
    return static_cast<float>(sum)/3;
}
int main()
{
    STUDENT s;
    s.get_data();
    s.display();
    float average=mark_avg(s);
    cout<<"Average marks of 3 tests:"<<average<<endl;
    return 0;
}
```

OUTPUT:

```
Enter the student's name:RAM
Enter the marks for three subjects:
Enter marks1:22
Enter marks2:23
Enter marks3:25
Student Name:RAM
Average marks of 3 tests:23.3333
```

PROGRAM 7:

Design, develop and execute a program in C++ based on the following requirements: An EMPLOYEE class containing data members & members functions: i) Data members: employee number (an integer), Employee_ Name (a string of characters), Basic_ Salary (in integer), All_ Allowances (an integer), Net_Salary (an integer). (ii) Member functions: To read the data of an employee, to calculate Net_Salary & to print the values of all the data members. (All_Allowances = 123% of Basic, Income Tax (IT) =30% of gross salary (=basic_ Salary_All_Allowances_IT).

```
#include<iostream>
#include<string>
using namespace std;
class EMPLOYEE
{
private:
   int employee_number;
   string employee_name;
   int basic_salary;
   int all_allowances;
   int net_salary;
public:
   void readData()
   {
      cout<<"Enter Employee Number:";
      cin>>employee_number;
```

```
        cout<<"Enter Employee Name:";
        cin.ignore();
        getline(cin,employee_name);
        cout<<"Enter Basic Salary:$";
        cin>>basic_salary;
    }
    void calculateNetSalary()
    {
        all_allowances=1.23*basic_salary;
        int gross_salary=basic_salary+all_allowances;
        int income_tax=0.3*gross_salary;
        net_salary=gross_salary-income_tax;
    }
    void printData()
    {
        cout<<"Employee Number:"<<employee_number<<endl;
        cout<<"Employee Name:"<<employee_name<<endl;
        cout<<"Basic Salary:$"<<basic_salary<<endl;
        cout<<"All Allowances:$"<<all_allowances<<endl;



        cout<<"Net Salary:$"<<net_salary<<endl;
    }
};
int main()
{
    EMPLOYEE emp;
    emp.readData();
    emp.calculateNetSalary();
```

```
    emp.printData();
    return 0;
}
```

OUTPUT:

```
Enter Employee Number:98
Enter Employee Name:Ramesh
Enter Basic Salary:$30000
Employee Number:98
Employee Name:Ramesh
Basic Salary:$30000
All Allowances:$36900
Net Salary:$46830
```

PROGRAM 8:

Write a C++ program to define class name FATHER & SON that holds the income respectively. Calculate & display total income of a family using Friend function

```
#include<iostream>
using namespace std;
class SON;
class FATHER
{
private:
    double income;
public:
    FATHER(double inc):income(inc){}
    friend double calculateTotalincome(const FATHER& father, const SON& son);
};
class SON
{
```

```
private:
   double income;
public:
   SON(double inc):income(inc){}
   friend double calculateTotalincome(const FATHER& father, const SON& son);
};
double calculateTotalincome(const FATHER& father, const SON& son)
{
   return father.income+son.income;
}
int main()
{
   double fatherincome,sonincome;
   cout<<"Enter Father's income:$";
   cin>>fatherincome;
   cout<<"Enter Son's income:$";
   cin>>sonincome;
   FATHER father(fatherincome);
   SON son(sonincome);
   double totalIncome=calculateTotalincome(father,son);
   cout<<"Total Income of the Family:$"<<totalIncome<<endl;
   return 0;
}
```

OUTPUT:

```
Enter Father's income:$20000
Enter Son's income:$60000
Total Income of the Family:$80000
```

PROGRAM 9:

Write a C++ program with different class related through multiple inheritance & demonstrate the use of different access specified by means of members variables & members functions.

```
#include<iostream>
using namespace std;
class Base1
{
public:
   int publicVar1;
   void publicFunc1()
   {
      cout<<"Base 1 Public Function"<<endl;
   }
protected:
   int protectedVar1;
   void protectedFunc1()
   {
      cout<<"Base 1 Protected Function"<<endl;
   }
   private:
   int privateVar1;
   void privateFunc1()
   {
      cout<<"Base 1 Private Function"<<endl;
   }
};
class Base2
{
```

```
public:
   int publicVar2;
   void publicFunc2()
   {
      cout<<"Base 2 Public Function"<<endl;
   }
protected:
   int protectedVar2;
   void protectedFunc2()
   {
      cout<<"Base 2 Protected Function"<<endl;
   }
   private:
   int privateVar2;
   void privateFunc2()
   {
      cout<<"Base 2 Private Function"<<endl;
   }
};
class Derived:public Base1, protected Base2
{
public:
   void accessBase1Members()
   {
      publicVar1=10;
      publicFunc1();
      cout<<"Public Variable 1:"<<endl;
      cout<<publicVar1<<endl;
      protectedVar1=20;
```

```
        protectedFunc1();
        cout<<"Protected Variable 1:"<<endl;
        cout<<protectedVar1<<endl;
    }
    void accessBase2Members()
    {
        publicVar2=30;
        publicFunc2();
        cout<<"Public Variable 2:"<<endl;
        cout<<publicVar2<<endl;
        protectedVar2=40;
        protectedFunc2();
        cout<<"Protected Variable 2:"<<endl;
        cout<<protectedVar2<<endl;
    }
};
int main()
{
    Derived d;

    d.accessBase1Members();
    d.accessBase2Members();
    return 0;
}
```

OUTPUT:

```
Base 1 Public Function
Public Variable 1:
10
Base 1 Protected Function
Protected Variable 1:
20
Base 2 Public Function
Public Variable 2:
30
Base 2 Protected Function
Protected Variable 2:
40
```

PROGRAM 10:

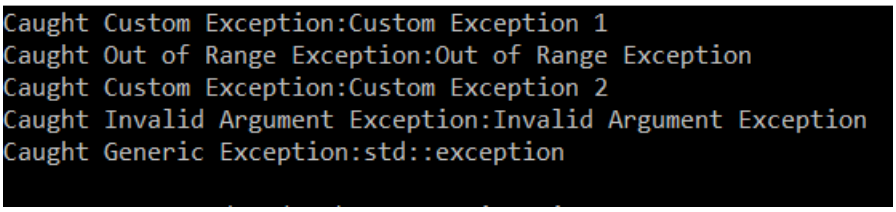
Write a C++ program to implement exception handling with minimum 5 exceptions classes including two built in exceptions.

```
#include<iostream>
#include<exception>
#include<stdexcept>
using namespace std;
class CustomException1: public exception
{
public:
   const char*what()const noexcept override
   {
      return "Custom Exception 1";
   }
};
class CustomException2: public exception
{
public:
   const char*what()const noexcept override
```

```
    {
        return "Custom Exception 2";
    }
};
int main()
{
    try
    {
        throw CustomException1();
    }
    catch(const CustomException1&e)
    {
        cout<<"Caught Custom Exception:"<<e.what()<<endl;
    }
     try
    {
        throw out_of_range("Out of Range Exception");
    }
    catch(const out_of_range&e)
    {
        cout<<"Caught Out of Range Exception:"<<e.what()<<endl;
    }
    catch(const exception&e)
    {
        cout<<"Caught Exception:"<<e.what()<<endl;
    }
    try
    {
        throw CustomException2();
```

```
    }
    catch(const CustomException2&e)
    {
       cout<<"Caught Custom Exception:"<<e.what()<<endl;
    }
     try
    {
       throw invalid_argument("Invalid Argument Exception");
    }
    catch(const invalid_argument&e)
    {
       cout<<"Caught Invalid Argument Exception:"<<e.what()<<endl;
    }
    catch(const exception&e)
    {
       cout<<"Caught Exception:"<<e.what()<<endl;
    }
    try
    {
       throw exception();
    }
    catch(const exception&e)
    {
       cout<<"Caught Generic Exception:"<<e.what()<<endl;
    }
    return 0;
}
```

OUTPUT:



PROGRAM 11:

Write a C++ program to calculate the volume of different geometric shapes like cube, cylinder and sphere using function overloading concept.

```
#include<iostream>
using namespace std;
const float pi=3.14;
float vol(float l)
{
return l*l*l;
}
float vol(float r, float h){
return (pi*r*r*h);
}
float vol(float l, float b, float h)
{
return (l*b*h);
}
int main(){
   float l,r,b,h,t;
      cout<<"Enter the Length of Cube:"<<endl;
      cin>>l;
```

```
    t=vol(l);
    cout<<"Volume of Cube:"<<t<<endl;
    cout<<"Enter the Radius & Height of Cylinder:"<<endl;
    cin>>r>>h;
    t=vol(r,h);
    cout<<"Volume of Cylinder:" <<t<<endl;
    cout<<"Enter the Length,Breadth & Height of Rectangle:"<<endl;
    cin>>l>>b>>h;
    t=vol(l,b,h);
    cout<<"Volume of Rectangle:"<<t;
    return 0;
}
```

OUTPUT: